Available online at www.sciencedirect.com
SCIENCE DIRECT®
ELSEVIER Computers and Mathematics with Applications 48 (2004) 1541–1548
An International Journal
computers & mathematics with applications
www.elsevier.com/locate/camwa

Symbolic Substitution Based Canonical Recoding Algorithms

T. IMAM AND M. KAYKOBAD
Department of Computer Science and Engineering
Bangladesh University of Engineering and Technology
Dhaka, Bangladesh
tasadduq@bttb.net.bd kaykobad@cse.buet.ac.bd

(Received October 2003; revised and accepted May 2004)

Abstract—Among the different algorithms that have been proposed for the conversion of a binary number to its canonical modified signed-digit representation, the Reitwiesner's algorithm has been the most cited and regarded as the most efficient one in the computing literature. In this paper, however, we present two new conversion algorithms that are computationally more efficient than the Reitwiesner's algorithm. Both of our algorithms use a lookup table and proceed using the symbolic substitution technique. An analysis of the computational efficiency of the proposed algorithms as determined by a computer simulation has also been presented. © 2004 Elsevier Ltd. All rights reserved.

Keywords—Canonical modified signed digit, Symbolic substitution table, Canonical recoding algorithm, Parallel processing, Substitution step.

1. INTRODUCTION

Redundant signed-digit number system has been proposed for the design of adder with limited carry-propagation arithmetic [1]. For any radix, $r \geq 2$, a redundant signed-digit integer number, $X = (x_{n-1}, \ldots, x_1, x_0)$, represented with n digits, has the algebraic value, $X = \sum_{i=0}^{n-1} x_i r^i$, where each digit x_i assumes its value in the digit set, $S = \{-\alpha, -\alpha+1, \ldots, -1, 0, 1, \ldots, \alpha-1, \alpha\}$ and maximum digit magnitude α must satisfy $\lceil (r-1)/2 \rceil \leq \alpha \leq r-1$. A *modified signed-digit (MSD)* number system is a specialization of the redundant signed-digit number system with $\alpha = 1$ and $r = 2$. The representation of a number in the signed-digit system, however, is not unique. The modified signed-digit representation is called *canonical* if it contains no consecutive nonzero digits [2]. A *canonical modified signed-digit (CMSD)* number system gives the flexibility of unique representation of numbers, as well as the capability of carry-free arithmetic operations. So, a CMSD system has found special interest in the area of parallel arithmetic processing [3–5]. MSD and CMSD systems, however, are ternary number systems involving the use of three symbols: 0, 1, and -1. The conventional computer architecture, on the other hand, is based on binary logic and the corresponding binary number system. Arithmetic operations involving binary

The authors would like to thank anonymous referees and Professor M.A. Karim, Dean of Engineering of the City College of the City University of New York, for suggesting several improvements.

0898-1221/04/$ - see front matter Typeset by AMS-TEX
doi:10.1016/j.camwa.2004.05.008

numbers generate carry and, therefore, to take advantage of fast, parallel, carry-free arithmetic operations of CMSD system based devices, conversion of binary numbers to corresponding CMSD numbers is necessary.

Reitwiesner's algorithm [2] has been the most cited one [6–8] in this regard. The algorithm can construct the unique canonical signed-digit vector of a number X based on a symbolic substitution table. But, for a number with n digits in the binary expansion, the algorithm requires n parallel steps to compute the final CMSD representation. We present in this paper two new conversion algorithms. One of these algorithms can determine the CMSD representation in just $\lfloor n/2 \rfloor + 1$ parallel steps. As for the other algorithm, although it requires a larger and complex symbolic substitution table, it can perform the conversion in just one step and moreover, it can start the conversion from any direction: left to right or right to left.

2. REITWIESNER'S ALGORITHM

Reitwiesner's algorithm requires that the binary expansion of a number x is padded with an initial zero. The algorithm computes the signed-digit representation y starting from the least significant digit and proceeding to the left. Initially, an auxiliary binary variable c_0 is set to 0, and subsequently, the binary expansion of x is scanned. At each parallel step, the canonically recoded digit y_i and the next value of the auxiliary binary variable c_{i+1}, for $i = 0 \ldots, n-1$, are calculated using the values of x_i, x_{i+1}, and c_i according to a conversion and complementary conversion table as shown in Table 1 [2,3]. The complete CMSD representation is obtained after n parallel steps. For convenience, we have, throughout this document, used the symbol $\bar{1}$ to denote -1.

Table 1. Reitwiesner's conversion table and corresponding complementary conversion table.

c_i	x_{i+1}	x_i	c_{i+1}	y_iV
0	0	0	0	0
0	0	1	0	1
0	1	0	0	0
0	1	1	1	$\bar{1}$
1	0	0	0	1
1	0	1	1	0
1	1	0	1	$\bar{1}$
1	1	1	1	0

c_i	x_{i+1}	x_i	c_{i+1}	y_i
0	0	0	0	0
0	0	$\bar{1}$	0	$\bar{1}$
0	$\bar{1}$	0	0	0
0	$\bar{1}$	$\bar{1}$	$\bar{1}$	1
$\bar{1}$	0	0	0	$\bar{1}$
$\bar{1}$	0	$\bar{1}$	$\bar{1}$	0
$\bar{1}$	$\bar{1}$	0	$\bar{1}$	1
$\bar{1}$	$\bar{1}$	$\bar{1}$	$\bar{1}$	0

3. IMPROVED ALGORITHM 1

We present here a new conversion algorithm that requires $\lfloor n/2 \rfloor + 1$ parallel steps. One highlighting feature of this algorithm is that it can convert binary numbers represented in both signed 2's complement form and unsigned form to their correct representation in CMSD. Moreover, it can convert any given MSD number to CMSD representation. The corresponding algorithm and symbolic substitution table has been shown in Algorithm 3.1 and Table 2

Algorithm 3.1. MSD or Binary To CMSD Conversion

Let $X = X_{n-1}, \ldots, X_1, X_0$ *be a given MSD or unsigned binary or 2's complement binary number,* $C = C_{n+1}, \ldots, C_1, C_0$ *be an auxiliary binary variable used in symbolic substitution process,* $Y = Y_{n+1}, \ldots, Y_1, Y_0$ *be the output CMSD number and* $CT = CT_{n+1}, \ldots, CT_1, CT_0$ be the temporary auxiliary binary variable generated in each symbolic substitution step.

Set $C = 0$

if input is in 2's complement Binary representation **then**
  Pad 2 'MSB' digits at Left
else
  Pad 2 '0' digits at Left
end if
for *stepcount* = 1 to $\lfloor n/2 \rfloor + 1$ STEP 1 **do**
  *(
  for $i = 0$ to $n+1$ STEP 2 **do**
    Set $Y_{i+1}Y_i$ and $CT_{i+3}CT_{i+2}$ based on $X_{i+1}X_i$ and $C_{i+1}C_i$ according to the symbolic substitution table (Table 2). If there is an entry for further check(**CheckCond.1** or **Check Cond.2**), then perform the substitution based on X_{i+2} and C_{i+2} according to the corresponding Check-condition table.
  end for)*
  Set $X = Y$, Set $C = CT$
end for
The derived $Y = Y_{n+1}, \ldots, Y_1, Y_0$ *is the output CMSD number*
NB: The *-marked code runs in parallel for all digits

The demonstration of the algorithm has been shown in Demonstrations 3.1–3.3.

Table 2. Symbolic substitution table to convert any MSD and binary number to CMSD in $(\lfloor n/2 \rfloor + 1)$ steps

Bits $(X_{i+1}X_i)$	**Check:**$C_{i+1}C_i$ **Outputs:** $(Y_{i+1}Y_i, CT_{i+3}CT_{i+2})$		
	00	**01**	**0$\bar{1}$**
00	00, 00	01, 00	0$\bar{1}$, 00
01	01, 00	**CheckCond.1**	00, 00
0$\bar{1}$	0$\bar{1}$, 00	00, 00	**CheckCond.2**
10	**CheckCond.1**	0$\bar{1}$, 01	01, 00
$\bar{1}\bar{1}$	**CheckCond.2**	0$\bar{1}$, 00	01, 0$\bar{1}$
11	0$\bar{1}$, 01	00, 01	**CheckCond.1**
$\bar{1}\bar{1}$	01, 0$\bar{1}$	**CheckCond.2**	00, 0$\bar{1}$
1$\bar{1}$	01, 00	**CheckCond.1**	00, 00
$\bar{1}$1	0$\bar{1}$, 00	00, 00	**CheckCond.2**

Note: **CheckCond.1** and **CheckCond.2** imply further conditions as have been illustrated in Tables 3 and 4

Table 3. **CheckCond.1** for Table 2.

Bits $(X_{i+2}C_{i+2})$	Outputs: $Y_{i+1}Y_i, CT_{i+3}CT_{i+2}$
00 or 11 or 1$\bar{1}$ or $\bar{1}$1 or $\bar{1}\bar{1}$	10, 00
01 or 10	$\bar{1}$0, 01
0$\bar{1}$ or $\bar{1}$0	$\bar{1}$0, 01

Table 4. **CheckCond.2** for Table 2.

Bits $(X_{i+2}C_{i+2})$	Outputs: $Y_{i+1}Y_i, CT_{i+3}CT_{i+2}$
00 or 11 or 1$\bar{1}$ or $\bar{1}$1 or $\bar{1}\bar{1}$	$\bar{1}$0, 00
01 or 10	10, 0$\bar{1}$
0$\bar{1}$ or $\bar{1}$0	10, 0$\bar{1}$

Demonstration 3.1

Given number: 7(4 *bit unsigned binary*) = $(0\,1\,1\,1)$.
After padding 2 '0' *bits at Left:* $X = (0\,0\,0\,1\,1\,1)$.
Initial auxiliary variable $C = (0\,0\,0\,0\,0\,0)$
STEP 1: $Y = (0\,0\,0\,0\,1\,0\,\bar{1})$, $CT = (0\,0\,0\,1\,0\,0)$
STEP 2: $Y = (0\,0\,0\,1\,0\,0\,\bar{1})$, $CT = (0\,0\,0\,0\,0\,0)$
STEP 3: $Y = (0\,0\,0\,1\,0\,0\,\bar{1})$, $CT = (0\,0\,0\,0\,0\,0)$
The CMSD Output: $(0\,0\,0\,1\,0\,0\,\bar{1}) = 7$.

Demonstration 3.2

Given number: -9(5 *bit 2's complement binary*) = $(1\,0\,1\,1\,1)$.
Since Input is 2's Complement Binary, Padding 2 *'MSB' bits at Left:* $X = (1\,1\,1\,0\,1\,1\,1)$.
Initial auxiliary variable$C = (0\,0\,0\,0\,0\,0\,0)$
STEP 1: $Y = (0\,0\,\bar{1}\,0\,1\,0\,\bar{1})$, $CT = (0\,0\,0\,0\,1\,0\,0)$
STEP 2: $Y = (0\,0\,\bar{1}\,\bar{1}\,0\,0\,\bar{1})$, $CT = (0\,0\,1\,0\,0\,0\,0)$
STEP 3: $Y = (0\,0\,0\,\bar{1}\,0\,0\,\bar{1})$, $CT = (0\,0\,0\,0\,0\,0\,0)$
The CMSD Output: $(0\,0\,0\,\bar{1}\,0\,0\,\bar{1}) = -9$.

Demonstration 3.3

Given MSD number: -14(4bit MSD) = $(\bar{1}\,\bar{1}\,\bar{1}\,0)$.
After padding 2 '0' bits at Left: $X = (0\,0\,\bar{1}\,\bar{1}\,\bar{1}\,0)$.
Initial auxiliary variable $C = (0\,0\,0\,0\,0\,0)$
STEP 1: $Y = (0\,0\,0\,1\,1\,0)$, $CT = (0\,\bar{1}\,0\,\bar{1}\,0\,0)$
STEP 2: $Y = (0\,\bar{1}\,0\,0\,1\,0)$, $CT = (0\,0\,0\,0\,0\,0)$
STEP 3: $Y = (0\,\bar{1}\,0\,0\,1\,0)$, $CT = (0\,0\,0\,0\,0\,0)$
The CMSD Output: $(0\,\bar{1}\,0\,0\,1\,0) = -14$.

4. OPTIMIZATION OF THE IMPROVED ALGORITHM

The presented algorithm can be further improved by stopping the substitution process as soon as the temporary auxiliary variable CT reaches the value of all zero. To demonstrate the optimization, a computer program to simulate the conversion process was developed. For a given bit-size, the program could generate all the possible combinations of MSD numbers for that particular bit-size. For each of these MSD numbers, the number of parallel steps that are necessary to convert the number to CMSD representation was recorded. From these data, the cumulative percentage of the total possible combinations that are completely converted to CMSD representation within a particular number of steps was calculated. The result of the analysis is as shown in the graph of Figure 1. It can be seen that most of the MSD numbers (more than 90%) are converted to corresponding CMSD representation in many fewer steps than the specified maximum of $(\lfloor n/2 \rfloor + 1)$ parallel processing steps. Hence, for any MSD or binary number generated at random, our algorithm with optimization will show superiority over Reitwiesner's algorithm.

5. IMPROVED ALGORITHM 2

Here we present another algorithm that works only for binary numbers. But the exciting feature is that it can perform the conversion of a binary number to its CMSD equivalent in just one step. Also, the conversion process can begin from either side: left to right or right to left, thus implying the total parallelism. The algorithm is as shown in Algorithm 5.2

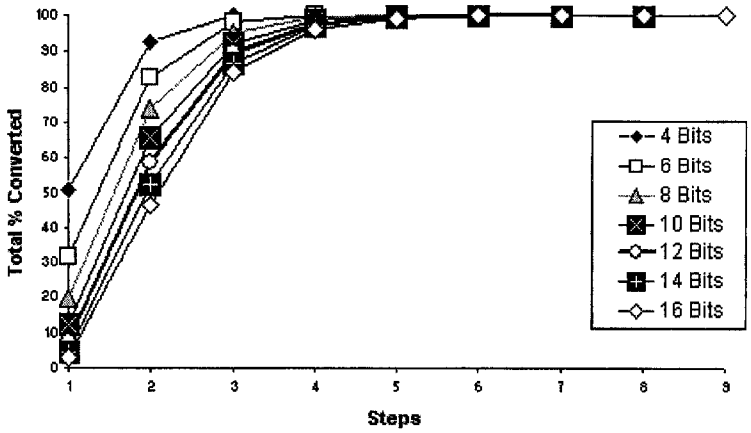

Figure 1. Graph of total of the percentage converted vs. required number of steps.

Algorithm 5.1. Search for a pattern like (10)*11. Used in Algorithm 5.2

Procedure *FindPattern(i)*
if there is a pattern like (10)*11 starting from bit position i **then**
Return True
else
Return False.
end if
End Procedure

Algorithm 5.2. The conversion algorithm

Procedure *DoConversion*
Let $X = X_{n-1}, \ldots, X_1, X_0$ *be a given unsigned binary or 2's complement binary number and* n *is even. Let* $Y = Y_{n+1}, \ldots, Y_1, Y_0$ *be the output CMSD number.*
Pad 2 0's at right
if X is a 2's complement binary number **then**
Sign extend X by 1 bit by padding the MSB bit 1 times at left
else
Pad 2 0's at left
endif
for i = n downto 0
switch $(X_i X_{i-1})$
case 00 or 11: $Y_i = 0$
case 01:
switch $(X_{i+1} X_{i-2})$
case 01: $Y_i = 1$
case 11: $Y_i = \bar{1}$
case 00:
if $(FindPattern(i-1))$ **then** $Y_i = 1$ **else** $Y_i = 0$
case 10:
if $(FindPattern(i-1))$ **then** $Y_i = \bar{1}$ **else** $Y_i = 0$
case 10:

switch $(X_{i+1}X_{i-2})$
  case 00: $Y_i = 1$
  case 10: $Y_i = \bar{1}$
  case 01:
    if $(FindPattern(i))$ **then** $Y_i = 0$ **else** $Y_i = 1$
  case 11:
    if $(FindPattern(i))$ **then** $Y_i = 0$ **else** $Y_i = \bar{1}$
End Procedure

The complete algorithm consists of two procedures. The first procedure, given a bit position i, searches for a pattern like (10)*11, i.e., patterns like 1011 or 101011 or 10101011 or so on, starting from the bit position i,. It returns true if such a pattern is found. The second procedure comprises the main conversion algorithm. Initially, it pads 2 0 bits referred to as X_{-1} and X_{-2} at the right of the binary expansion of the number to be converted. Depending on the representation, it also pads the MSB bit or 2 '0' bits at left. Then, for each bit position i, the algorithm checks X_i and X_{i-1} to determine the final digit Y_i, for $i = 0, \ldots, n-1$. If $X_iX_{i-1} = 01$ or 10, the algorithm checks the values of X_{i+1} and X_{i+1} also. The determination of the output digits are done in parallel for all bit-positions. Thus, the full conversion is performed in just one step. The demonstration of the algorithm is as shown in Demonstrations 5.1 and 5.2.

Demonstration 5.1.

Given number: 43 (6 bit unsigned binary) = (101011).
After padding 2 '0' bits at Right = (1010 1100).
Sine Input is unsigned Binary pad 2 '0' bits at Left : $X = (0010101100)$
For bit position 6: $X_6X_5 = 01$: $X_7X_4 = 00$: *FoundPattern(5) = True :* $Y_6 = 1$
For bit position 5: $X_5X_4 = 10$: $X_6X_3 = 01$: *FoundPattern(5) = True :* $Y_5 = 0$
For bit position 4: $X_4X_3 = 01$: X_5X_2=*10* : *FoundPattern(3) = True :* $Y_4 = \bar{1}$
For bit position 3: $X_3X_2 = 10$: X_4X_1=*01* : *FoundPattern(3) = True :* $Y_3 = 0$
For bit position 2: $X_2X_1 = 01$: X_3X_0=*11* : $Y_2 = \bar{1}$
For bit position 1: $X_1X_0 = 11$: $Y_1 = 0$
For bit position 0: $X_0X_{-1} = 10$: $X_1X_{-2} = 10$: $Y_0 = \bar{1}$
The CMSD Output: $(10\bar{1}0\bar{1}0\bar{1}) = 43$.

Demonstration 5.2.

Given number: -21 *(6 bit 2's complement binary)* = (101011).
After padding 2 '0' bits at Right = (10101100).
Sine Input is 2's complement Binary pad the MSB bit once at Left : $X = (110101100)$
For bit position 5: $X_5X_4 = 10$: $X_6X_3 = 11$: *FoundPattern(5) = True :* $Y_5 = 0$
For bit position 4: $X_4X_3 = 01$: $X_5X_2 = 10$: *FoundPattern(3) =T rue :* $Y_4 = \bar{1}$
For bit position 3: $X_3X_2 = 10$: $X_4X_1 = 01$: *FoundPattern(3) = True :* $Y_3 = 0$
For bit position 2: $X_2X_1 = 01$: $X_3X_0 = 11$: $Y_2 = \bar{1}$
For bit position 1: $X_1X_0 = 11$: $Y_1 = 0$
For bit position 0: $X_0X_{-1} = 10$: $X_1X_{-2} = 10$: $Y_0 = \bar{1}$
The CMSD Output: $(0\bar{1}0\bar{1}0\ \bar{1}) = -21$.

Although the algorithm can convert a binary representation to corresponding CMSD in one step, it has the disadvantage of checking for (10)*11 pattern in the representation. This checking implies that for any bit position i of the binary number to be converted, the algorithm may require the values of bit positions $i+1, \ldots, i-n+1$ to make the conversion decision. Thus,

the symbolic substitution table has to contain entries for all bit positions in the check condition. In other words, if the bit-size of the binary number to be converted is n, then the symbolic substitution table has to contain column entries for bit positions $X_{n+1}, \ldots, X_0$. However, not all the permutations of these bits are necessary for inclusion as conversion rules i.e., row entries in the symbolic substitution table. If the conditions specified in the algorithm are co-opted in the symbolic substitution table, then for the conversion of an n bit binary number, we need a symbolic substitution table with $(n+1)$ column entries for check conditions, and $4 * (n-1)$ row entries for conversion rules. Thus, for bit-size of 4, 6, and 8, we need only 12, 20, and 28 conversion rules with 5, 7, and 9 check conditions. The dependency of the symbolic substitution table on the bit size of the binary number to be converted may appear to be a disadvantage. Also, the inability to make the symbolic substitution table generalized to perform a conversion of a binary number of any size may appear to be a flaw. However, at the cost of some extra entries and nongeneralization, the advantage of performing the conversion in just one step may outweigh these disadvantages in many parallel processing situations.

An example of a symbolic substitution table, that can be used to convert a 4-bit binary number, is shown in Table 5. The blank (-) entries in the table indicate a do not care condition for that bit position.

Table 5. Symbolic substitution table for converting any 4-bit binary number to CMSD representation in one step.

X_{i+1}	X_i	X_{i-1}	X_{i-2}	X_{i-3}	Y_i
–	0	0	–	–	0
–	1	1	–	–	0
0	0	1	1	–	1
0	0	1	0	–	0
1	0	1	1	–	$\bar{1}$
1	0	1	0	–	0
0	1	0	1	1	0
0	1	0	1	0	1
0	1	0	0	–	1
1	1	0	1	1	0
1	1	0	1	0	$\bar{1}$
1	1	0	0	–	$\bar{1}$

6. A COMPARISON OF DIFFERENT SCHEMES

We have introduced two new conversion algorithms and corresponding symbolic substitution tables. The first algorithm uses a generalized symbolic substitution table to perform the conversion of binary or MSD numbers of any size. It can handle binary numbers represented in 2's complement form as well. Moreover, if optimization is introduced, the algorithm can process most of the MSD numbers in many fewer steps than that of the nonoptimized version. The second improved algorithm requires a fixed symbolic substitution table that cannot be generalized to handle variable sized binary numbers and requires more entries than other schemes. However, the algorithm can perform the conversion in just one step. Both of these algorithms, even though requiring larger substitution tables than Reitwiesner's algorithm, are computationaly more efficient in terms of the required number of parallel conversion steps. While Reitwiesner's algorithm requires n conversion steps for any n bit binary number, the first proposed algorithm requires $\lfloor n/2 \rfloor + 1$ steps, whereas the second algorithm requires just one parallel processing step.

7. CONCLUSION

The paper has made an effort to improve the Reitwiesner's algorithm in terms of the required

number of parallel substitution steps. In situations where we need a unique conversion method for performing the conversion of both unsigned and two's complement binary and MSD numbers, the first algorithm as well as its optimized version will prove to be superior over other algorithms. In other situations, where we can provide the required memory storage for the symbolic substitution table and where the number to be converted is represented in binary notation of a predefined fixed size, the second algorithm will give the fastest conversion rate possible.

REFERENCES

1. A. Avizienis, Signed-digit number representations for fast parallel arithmetic, *IRE Trans. Electronic Computers* **10**, 389–400, (1961).
2. G. Reitwiesner, Binary arithmetic, *Advances in Computers* **1**, 231–308, (1960).
3. M. Alam, A. Awwal and A. Cherry, Opto-electronic symbolic substitution based canonical modified signed-digit arithmetic, *Optics and Laser Technology* **29**, 151–157, (1997).
4. A. Awwal and M. Karim, *Optical Computing: An Introduction*, John Wiley & Sons, New York, (1992).
5. J. Ahmed and A. Awwal, Polarization-encoded optical shadow-casting arithmetic-logic-unit design: Separate and simultaneous output generation, *Applied Optics* **31**, 5622–5631, (1992).
6. O. Egecioglu and C.K. Koc, Exponentiation using canonical recoding, *Theoretical Computer Science* **129** (2), 407–417, (1994).
7. M. Joye and S.-M. Yen, Optimal left-to-right binary signed-digit recoding, *IEEE Transactions on Computers* **49** (7), 740–748, (2000).
8. C.K. Koc, Parallel canonical recoding, *Electronics Letters* **32** (22), 2063–2065, (1996).